Math. Struct. in Comp. Science: page 1 of 21. © Cambridge University Press 2016
doi:10.1017/S0960129515000481

Well-founded coalgebras, revisited

JEAN-BAPTISTE JEANNIN[†], DEXTER KOZEN[†]
and ALEXANDRA SILVA[‡]

[†]*Computer Science, Cornell University, Ithaca, New York 14853-7501, U.S.A.*
Email: kozen@cs.cornell.edu
[‡]*Intelligent Systems, Radboud University Nijmegen, Postbus 9010, 6500 GL Nijmegen, the Netherlands*
Email: alexandra@cs.ru.nl

Received 6 March 2013; Revised 24 March 2015

Theoretical models of recursion schemes have been well studied under the names well-founded coalgebras, recursive coalgebras, corecursive algebras and Elgot algebras. Much of this work focuses on conditions ensuring unique or canonical solutions, e.g. when the coalgebra is well founded.

If the coalgebra is not well founded, then there can be multiple solutions. The standard semantics of recursive programs gives a particular solution, typically the least fixpoint of a certain monotone map on a domain whose least element is the totally undefined function; but this solution may not be the desired one. We have recently proposed programming language constructs to allow the specification of alternative solutions and methods to compute them. We have implemented these new constructs as an extension of OCaml.

In this paper, we prove some theoretical results characterizing well-founded coalgebras, along with several examples for which this extension is useful. We also give several examples that are not well founded but still have a desired solution. In each case, the function would diverge under the standard semantics of recursion, but can be specified and computed with the programming language constructs we have proposed.

1. Introduction

Infinite coinductive datatypes and functions on them offer interesting challenges in the design of programming languages. While most programmers feel comfortable with inductive datatypes, coinductive datatypes are often considered difficult to handle. Many programming languages do not even provide constructs to define them. OCaml offers the possibility of defining coinductive datatypes, but the means to define recursive functions on them are limited. Often the obvious definitions do not halt or provide the wrong solution.

Theoretical models of recursion schemes have been well studied under the names well-founded coalgebras, recursive coalgebras (Adámek *et al.* 2007), corecursive algebras (Capretta *et al.* 2009) and Elgot algebras (Adámek *et al.* 2006). Much of this work focuses on conditions ensuring unique or canonical solutions, e.g. when the coalgebra is well founded.

A prototypical example of a function that fits the well-founded scheme is mergesort. Given a list, we can sort it by dividing it into identical pieces, sorting the smaller lists, then

merging the resulting sorted lists. The base case is the empty list or the list containing a single element. As with most recursive functions, the scheme of definition is: given an argument, check if it is the base case; if not, prepare the arguments for the recursive calls, recursively apply the function, then combine the results of the recursive calls into the final result. For mergesort, this scheme is illustrated in the following diagram:

$$\begin{array}{ccc} A^* & \xrightarrow{\quad\mathsf{mergesort}\quad} & A^* \\ {\scriptstyle\gamma}\downarrow & & \uparrow{\scriptstyle\alpha} \\ A^* + A^* \times A^* & \xrightarrow[\mathsf{id}_{A^*} + \mathsf{mergesort} \times \mathsf{mergesort}]{} & A^* + A^* \times A^* \end{array}.$$

The function γ checks whether the list is empty or a singleton, otherwise divides it in two lists of roughly equal size.

$$\gamma(\ell) = \iota_0(\ell), \quad \ell = [\,] \text{ or } \ell = [a],$$
$$\gamma([a_1; \ldots; a_n]) = \iota_1([a_1; \ldots; a_{\lfloor n/2 \rfloor}], [a_{\lfloor n/2 \rfloor + 1}; \ldots; a_n]), \quad n \geqslant 2.$$

Here ι_0 and ι_1 are the injections into the coproduct. After the function is applied recursively, the results of the recursive calls are combined by α, which merges the two sorted lists.

$$\alpha(\iota_0(\ell)) = \ell \qquad\qquad \alpha(\iota_1(\ell_1, \ell_2)) = \mathsf{merge}(\ell_1, \ell_2)$$

The merge function obeys a similar scheme:

$$\begin{array}{ccc} A^* \times A^* & \xrightarrow{\quad\mathsf{merge}\quad} & A^* \\ {\scriptstyle\gamma}\downarrow & & \uparrow{\scriptstyle\alpha} \\ A^* + A \times A^* \times A^* & \xrightarrow[\mathsf{id}_{A^*} + \mathsf{id}_A \times \mathsf{merge}]{} & A^* + A \times A^* \end{array},$$

where

$$\gamma([\,], \ell) = \gamma(\ell, [\,]) = \iota_0(\ell), \qquad\qquad \alpha(\iota_0(\ell)) = \ell$$
$$\gamma(a_1 :: \ell_1, a_2 :: \ell_2) = \begin{cases} \iota_1(a_1, \ell_1, a_2 :: \ell_2) & \text{if } a_1 \leqslant a_2 \\ \iota_1(a_2, a_1 :: \ell_1, \ell_2) & \text{if } a_1 > a_2 \end{cases} \qquad\qquad \alpha(\iota_1(a, \ell)) = a :: \ell.$$

The fact that these functions are well-defined and unique follows from the theory of recursive coalgebras (Adámek *et al.* 2007).

Abstractly, these definitional schemes are of the form

$$\begin{array}{ccc} C & \xrightarrow{\quad h\quad} & A \\ {\scriptstyle\gamma}\downarrow & & \uparrow{\scriptstyle\alpha} \\ FC & \xrightarrow[Fh]{} & FA \end{array}, \tag{1}$$

where F is usually a polynomial functor on Set and (C, γ) and (A, α) are a coalgebra and an algebra, respectively, for the functor F. The function h being defined is called an *F-coalgebra-algebra morphism.*

The standard semantics of recursion, as provided by all modern programming languages, provides a means of expressing and computing the unique solution of (1), provided the coalgebra C is well founded; that is, provided there is a basis to the recursion. However, the diagram (1) can act as a valid definitional scheme even when C is not well founded. This observation was the starting point of our work on new program constructs for functions defined by such definitional schemes when C is not well founded (Jeannin *et al.* 2012, 2013). In the non-well-founded case, there can be multiple solutions. The standard semantics of recursive programs gives a particular solution, typically the least fixpoint of a certain monotone map on a domain whose least element is the totally undefined function, which in the non-well-founded case may not be the desired one. In Jeannin *et al.* (2012, 2013), we introduced new programming language constructs to allow the programmer to specify and compute a desired alternative solution by solving a set of equations determined by (1).

In the course of our study, we also proved some theoretical results that are connected to existing research (Adámek *et al.* 2007). The setting in our paper and the one of Adámek *et al.* (2007) is different though: on the one hand, in Adámek *et al.* (2007) the functor is required to be finitary, which we do not in the present paper; on the other hand, we do require functors to be polynomial (albeit multi-sorted) which they do not. In this paper, we provide some examples where our extension is useful. The simplest example of this is the case of mutually recursive definitions. For example, consider the even and odd predicates on natural numbers. In an ML-style language, we would write:

```
let rec even n = if n = 0 then true else odd (n-1)
and odd n = if n = 0 then false else even (n-1)
```

Our results extend the results of Adámek *et al.* (2007) to several patterns of function definitions, including this one. Mutually recursive functions are also treated in (Adámek *et al.* 2006). We want to stress again that the results of Adámek *et al.* (2007) apply to a large class of recursive function definitions, but there appear to be cases that are not covered, at least not in any straightforward way, like the above one.

The main result of this paper is a theoretical result that relates to a result of Adámek *et al.* (2007). In particular, we consider multi-sorted polynomial functors instead of finitary functors. As we shall see, this provides an amenable framework for mutually recursive functions. Moreover, we clarify that the central notion in the theorem is well-foundedness and not finiteness. We show:

— Every F-coalgebra C contains a maximal well-founded subcoalgebra $\mathsf{wf}\, C$.
— If R is a final F-coalgebra, then $\mathsf{wf}\, R$ is the initial F-algebra.
— Let C be an F-coalgebra. The following are equivalent:
  – C is well founded; that is, $C = \mathsf{wf}\, C$.
  – There is a valid induction principle for C (defined precisely in Section 3.2).
  – There is a unique coalgebra morphism $C \to \mathsf{wf}\, R$.
  – There is a unique coalgebra-algebra morphism from C to any F-algebra.

Our constructions are based on the concept of *realizations*, a concrete representation of final coalgebras for a wide class of multi-sorted type signatures (Kozen 2011). Realizations go beyond ordinary polynomial functors on Set in that they handle infinite (countable or

uncountable) product and sum as well as total and partial functions. They also handle multi-sorted signatures in a more symmetric way, without relying on any Cartesian structure or parameterization as in Adámek *et al.* (2006). Our second contribution is a variety of well-founded and non-well-founded examples that illustrate the power and limitations of the theory.

The paper is organized as follows. In Section 2 we review the results of Kozen (2011) on realizations of coinductive types, which are essential to the understanding of our main theoretical results in Section 3. In Section 3 we give a new characterization of well-founded coalgebras in terms of realizations. In Section 4 we present several examples of well-founded applications. Some of these are already covered by the results of Adámek *et al.* (2007), but others, such as mutually recursive functions even/odd and the Ackermann function, are not. However, each of these exhibits some interesting or surprising characteristic that attests to the wide applicability of the theory. In Section 5 we present several non-well-founded examples, including an example of Capretta (2007) involving descending sequences of natural numbers and the semantics of alternating Turing machines and IND programs (Harel and Kozen 1984). These examples illustrate the usefulness of (1) as a definitional scheme even in the non-well-founded case. In Section 6 we briefly describe our experience with bringing these theoretical ideas to practical fruition in the form of new programming language constructs for specifying alternative solutions to (1). These practical results are reported more fully in Jeannin *et al.* (2012, 2013), but here we are able to put them in the proper theoretical context. We conclude in Section 7 with a discussion of related theoretical and practical results.

2. Realization of coinductive types

In the proof of Theorem 3.3, we make use of an explicit construction of final coalgebras from Kozen (2011). To make this paper self-contained, this section recalls the main definitions and results.

2.1. *Directed multigraphs*

Type signatures will be represented by directed multigraphs. A *directed multigraph* is a structure $G = (V, E, \mathsf{src}, \mathsf{tgt})$ with nodes V, edges E, and two maps $\mathsf{src}, \mathsf{tgt} : E \to V$ giving the source and target of each edge, respectively. We write $e : s \to t$ if $s = \mathsf{src}\, e$ and $t = \mathsf{tgt}\, e$. When specifying multigraphs, we will sometimes use the notation $s \xrightarrow{n} t$ for the metastatement, 'There are exactly n edges from s to t.'

A *path* is a finite alternating sequence of nodes and edges

$$s_0\, e_1\, s_1\, e_2\, s_2 \cdots s_{n-1}\, e_n\, s_n,$$

$n \geqslant 0$, such that $e_i : s_{i-1} \to s_i$, $1 \leqslant i \leqslant n$. These are the arrows of the free category generated by G. The *length* of a path is the number of edges. A path of length 0 is just a single node. The first and last nodes of a path p are denoted $\mathsf{src}\, p$ and $\mathsf{tgt}\, p$, respectively. As with edges, we write $p : s \to t$ if $s = \mathsf{src}\, p$ and $t = \mathsf{tgt}\, p$.

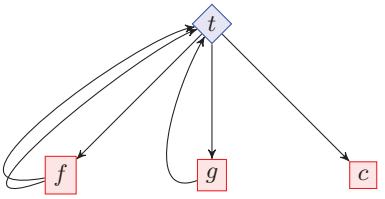


Fig. 1. A multigraph representing a single-sorted algebraic signature. Blue diamonds represent existential nodes and red squares universal nodes.

A *multigraph homomorphism* $\ell : G_1 \to G_2$ is a map $\ell : V_1 \to V_2$, $\ell : E_1 \to E_2$ such that if $e : s \to t$ then $\ell(e) : \ell(s) \to \ell(t)$. This lifts to a functor on the free categories generated by G_1 and G_2.

2.2. *Type signatures*

A *type signature* is a directed multigraph F along with a designation of each node of F as either *existential* or *universal*. The existential and universal nodes correspond respectively to coproduct and product constructors. The directed edges of the graph represent the corresponding destructors.

For example, consider an algebraic signature consisting of a binary function symbol f, a unary function symbol g, and a constant c. This would ordinarily be represented by the polynomial endofunctor $FX = X^2 + X + \mathbb{1}$, or in OCaml by

```
type t = F of t * t | G of t | C
```

We would represent this signature by a directed multigraph consisting of four nodes $\{t, f, g, c\}$, of which t is existential and f, g, c are universal, along with edges

$$t \xrightarrow{1} f \qquad t \xrightarrow{1} g \qquad t \xrightarrow{1} c \qquad f \xrightarrow{2} t \qquad g \xrightarrow{1} t.$$

The multigraph is illustrated in Figure 1.

2.3. *Coalgebras and realizations*

Let F be a type signature with nodes V. An *F-coalgebra* is a V-indexed collection of pairs (C_s, γ_s), where the C_s are sets and the γ_s are set functions

$$\gamma_s : C_s \to \begin{cases} \sum_{\mathsf{src}\, e = s} C_{\mathsf{tgt}\, e}, & \text{if } s \text{ is existential,} \\ \prod_{\mathsf{src}\, e = s} C_{\mathsf{tgt}\, e}, & \text{if } s \text{ is universal.} \end{cases}$$

A morphism of F-coalgebras is a V-indexed collection of set maps h_s that commute with the transition maps in the usual way:

$$\begin{array}{ccc} C_s & \xrightarrow{h_s} & D_s \\ {\scriptstyle \gamma_s}\downarrow & & \downarrow{\scriptstyle \xi_s} \\ FC_s & \xrightarrow[Fh_s]{} & FD_s \end{array}.$$

Similarly, an *F-algebra* is a V-indexed collection of pairs (A_s, α_s), where the A_s are sets and the α_s are set functions

$$\alpha_s : \left\{ \begin{array}{l} \sum_{\mathsf{src}\, e = s} A_{\mathsf{tgt}\, e}, \text{ if } s \text{ is existential,} \\ \prod_{\mathsf{src}\, e = s} A_{\mathsf{tgt}\, e}, \text{ if } s \text{ is universal} \end{array} \right\} \to A_s.$$

A morphism of F-algebras is a V-indexed collection of set maps h_s that commute with the transition maps in the usual way:

$$\begin{array}{ccc} A_s & \xrightarrow{h_s} & B_s \\ {\scriptstyle \alpha_s}\uparrow & & \uparrow{\scriptstyle \beta_s} \\ FA_s & \xrightarrow[Fh_s]{} & FB_s \end{array} . \tag{2}$$

These definitions correspond to the traditional definition of F-coalgebras and F-algebras for an endofunctor F on Set^V.

Coalgebras are equivalent to *realizations*. An *F-realization* is a directed multigraph G along with a multigraph homomorphism $\ell : G \to F$, called a *typing*, with the following properties.

— If $\ell(u)$ is existential, then there is exactly one edge of G with source u.
— If $\ell(u)$ is universal, then ℓ is a bijection between the edges of G with source u and the edges of F with source $\ell(u)$.

A homomorphism of F-realizations is a multigraph homomorphism that commutes with the typings.

Theorem 2.1 (Kozen 2011). The categories of F-coalgebras and F-realizations are equivalent (in the sense of Mac Lane (1971, Section IV.4)).

Briefly, one forms a realization from a coalgebra $(C_s, \gamma_s \mid s \in V)$ by connecting each existential element $u \in C_s$ to the unique v such that $\gamma_s(u) = \mathsf{in}_e(v)$ and each universal element $u \in C_t$ to all v forming the tuple $\gamma_t(u)$. See Kozen (2011) for a more detailed exposition.

2.4. *Final coalgebras*

Realizations allow a concrete construction of final coalgebras that is reminiscent of the Brzozowski derivative on sets of strings. Here, instead of strings, the derivative acts on certain sets of paths of the type signature.

Let F be a type signature. Construct a realization R, ℓ as follows. A node of R is a set A of finite paths in F such that

— A is nonempty and prefix-closed;
— all paths in A have the same first node, which we define to be $\ell(A)$;
— if p is a path in A of length n and $\mathsf{tgt}\, p$ is existential, then there is exactly one path of length $n+1$ in A extending p;
— if p is a path in A of length n and $\mathsf{tgt}\, p$ is universal, then all paths of length $n+1$ extending p are in A.

The edges of R are defined as follows. Let A be a set of paths in F and e an edge of F. Define the *Brzozowski derivative* of A with respect to e to be

$$D_e(A) = \{p \mid (\mathsf{src}\, e)\, e\, p \in A\},$$

the set of paths obtained by removing the initial edge e from paths in A that start with that edge. If A is a node of R and $D_e(A)$ is nonempty, we include exactly one edge

$$\langle A, e\rangle : A \to D_e(A),$$

in R and take $\ell(\langle A, e\rangle) = e$. It is readily verified that $\mathsf{tgt}\,\langle A, e\rangle = D_e(A)$ satisfies the four properties above required to be a node of R and that $\ell(D_e(A)) = \mathsf{tgt}\, e$, so ℓ is a typing.

Theorem 2.2 (Kozen 2011). The realization R, ℓ is final in the category of F-realizations. The corresponding F-coalgebra as constructed in Theorem 2.1 is final in the category of F-coalgebras.

3. Characterization of well-founded coalgebras

Well-founded coalgebras have a precise characterization in terms of their corresponding realizations: a coalgebra is *well founded* if and only if its corresponding realization is well founded as a graph; that is, if it has no infinite directed paths. The main theorem of Adámek *et al.* (2007) characterizes halting in terms of finiteness instead of well-foundedness, which by König's lemma is equivalent for the finitary functors considered in Adámek *et al.* (2007). However, one should stress that the essential property at play is really well-foundedness and not finiteness. In the following, we consider coalgebras for a wider class of functors, namely multi-sorted polynomial functors on Set^V, where V is a set of sorts, with infinite (countable and uncountable) product and sum, as well as total and partial functions. This is the same class of functors considered in Kozen (2011). From this point onwards, we will use F to refer to a multi-sorted polynomial functor. We will be making use of the fact that F-coalgebras and F-realizations are in one-to-one correspondence, and we will often use the corresponding type signature to represent the functor F.

When a recursive function is called on a well-founded argument, the solution is unique and the standard semantics will terminate. Theorem 3.3, which generalizes (Adámek *et al.* 2007) to the non-finitary case, characterizes the conditions under which this occurs.

The proof of Theorem 3.3 relies on some extra interesting facts which we also prove, namely that every F-coalgebra C contains a unique maximal well-founded subcoalgebra $\mathsf{wf}\, C$ and that if R is the final F-coalgebra, then $\mathsf{wf}\, R$ is the initial F-algebra.

3.1. *Well-founded coalgebras*

Let F be a functor, with corresponding type signature with nodes V. An *F-coalgebra-algebra morphism* is a V-indexed collection of set functions $h_s : C_s \to A_s$, where $(C_s, \gamma_s \mid s \in V)$ is an F-coalgebra and $(A_s, \alpha_s \mid s \in V)$ is an F-algebra in the sense of Section 2.3,

such that

$$h_s = \begin{cases} \alpha_s \circ [h_{\mathsf{tgt}\, e} \mid \mathsf{src}\, e = s] \circ \gamma_s, & \text{if } s \text{ is existential,} \\ \alpha_s \circ \langle h_{\mathsf{tgt}\, e} \mid \mathsf{src}\, e = s \rangle \circ \gamma_s, & \text{if } s \text{ is universal,} \end{cases} \tag{3}$$

where

$$[h_t \mid t \in I] : \sum_{t \in I} C_t \to \sum_{t \in I} A_t \qquad \langle h_t \mid t \in I \rangle : \prod_{t \in I} C_t \to \prod_{t \in I} A_t$$

are universal arrows for the coproduct and product, respectively, in Set. This condition can be represented pictorially by the abbreviated diagram

$$\begin{array}{ccc} C & \xrightarrow{\;h\;} & A \\ {\scriptstyle\gamma}\downarrow & & \uparrow{\scriptstyle\alpha} \\ FC & \xrightarrow[\;Fh\;]{} & FA \end{array} \tag{4}$$

If C is a realization instead of a coalgebra, the condition (3) becomes

$$h_s(u) = \begin{cases} \alpha_s(\mathsf{in}_e(h_{\mathsf{tgt}\, e}(v))), & \text{if } s \text{ is existential,} \\ \alpha_s(h_{\mathsf{tgt}\, e}(v_e) \mid \mathsf{src}\, e = s), & \text{if } s \text{ is universal,} \end{cases}$$

where if s is existential, (u, v) is the unique edge of C with source u and $e = \ell(u, v)$; and if s is universal, (u, v_e) are the edges of C with source u and $e = \ell(u, v_e)$.

An F-realization $G = (U, E, \mathsf{src}, \mathsf{tgt}, \ell)$ is *well founded* if all directed E-paths are finite. An F-coalgebra is *well founded* if its corresponding F-realization is.

Lemma 3.1. Every F-coalgebra contains a unique maximal well-founded subcoalgebra.

Proof. Equivalently, every F-realization $G = (U, E, \mathsf{src}, \mathsf{tgt}, \ell)$ contains a unique maximal well-founded F-subrealization $\mathsf{wf}\, G$. The nodes of $\mathsf{wf}\, G$ are the nodes of G from which there are no infinite directed E-paths. The graph $\mathsf{wf}\, G$ is the induced subgraph on this set of nodes. Equivalently, the set of nodes of $\mathsf{wf}\, G$ is the smallest set of nodes A of G satisfying the closure condition: if all E-successors of s are in A, then $s \in A$.

The subgraph $\mathsf{wf}\, G$ is a subrealization because if $s \in \mathsf{wf}\, G$, then all E-successors of s are in $\mathsf{wf}\, G$: if there are no infinite directed paths out of s, then there cannot be an infinite directed path out of any successor of s. Moreover, $\mathsf{wf}\, G$ is the unique maximal well-founded subrealization, because any node s not in $\mathsf{wf}\, G$ must be the starting point of an infinite directed path by definition, therefore G is not well founded below s. □

Lemma 3.2. Let $R = (V, E, \mathsf{src}, \mathsf{tgt}, \ell)$ be the final F-realization. Then $\mathsf{wf}\, R$ is an F-algebra.

Proof. By Lambek's lemma (Lambek 1968), the structure map $(\gamma_s \mid s \in V)$ of the final F-coalgebra corresponding to R is invertible, thus forms an F-algebra. Translating back to the realization R, this means that

— for every edge $e \in E$ such that $\mathsf{src}\, e$ is existential and every node v of R with $\ell(v) = \mathsf{tgt}\, e$, there exists a unique node u and edge d of R such that $\mathsf{src}\, d = u$, $\mathsf{tgt}\, d = v$, and $\ell(d) = e$; and
— for every universal node $s \in V$ and tuple $(v_e \mid \mathsf{src}\, e = s)$ of nodes of R such that $\ell(v_e) = \mathsf{tgt}\, e$, there exist a unique node u and tuple of edges $(d_e \mid \mathsf{src}\, e = s)$ of R such that $\mathsf{src}\, d_e = u$, $\mathsf{tgt}\, d_e = v_e$, and $\ell(d_e) = e$.

The existence and uniqueness of u in the above two cases assert the closure of R under the algebraic operations. The subrealization $\mathsf{wf}\, R$ is closed under these operations, because any node all of whose immediate E-successors are in $\mathsf{wf}\, R$ is also in $\mathsf{wf}\, R$, therefore $\mathsf{wf}\, R$ is a subalgebra of R. □

We will show in Corollary 3.4 that $\mathsf{wf}\, R$ is in fact the initial F-algebra (up to isomorphism). To show initiality, we need to show that there is a unique F-algebra morphism to any other F-algebra. This will follow as a special case of Theorem 3.3(iv) below.

3.2. *Induction principle*

The well-founded part of a realization G can be expressed in the modal μ-calculus as $\mathsf{wf}\, G = \mu X.\Box X$, where the modality $\Box$ is interpreted in G by the E-successor relation $E(x) = \{\mathsf{tgt}\, e \mid e \in E,\ \mathsf{src}\, e = x\}$; that is, the modal formula $\Box P$ holds of x if P holds of all E-successors of x. Thus, G is well founded if $\mu X.\Box X$ is universally valid in G.

The *induction principle* for a well-founded realization $G = (V, E, \mathsf{src}, \mathsf{tgt}, \ell)$ is:

$$\frac{\forall x\ (\forall y \in E(x)\ P(y)) \to P(x)}{\forall x\ P(x)}, \tag{5}$$

or more concisely,

$$\frac{\Box P \to P}{P}.$$

As we argue in Theorem 3.3, this rule is sound if and only if G is well founded.

3.3. *Main theorem*

We are now ready to state and prove our main theorem. We include point (v) to align with (Adámek *et al.* 2007, Theorem 3.8), although it is not really needed for our work. We give the definition of a *parameterized coalgebra-algebra morphism* in the context of the proof below.

Theorem 3.3. Let (C, γ) be an F-coalgebra and let R be the final F-coalgebra. The following are equivalent:

i. C is well founded; that is, $C = \mathsf{wf}\, C$.
ii. The induction principle (5) is valid for C.
iii. There is a unique coalgebra morphism $C \to \mathsf{wf}\, R$.
iv. There is a unique coalgebra-algebra morphism from C to any F-algebra.

v. There is a unique parameterized coalgebra-algebra morphism from C to any F-algebra.

Proof. The equivalence of (i) and (ii) is a fundamental property of relational algebra: a relation R is well founded if and only if the following induction principle holds. To show that $P(x)$ holds for all x it suffices to show that if x is an element of X and $P(y)$ holds for all y such that yRx, then $P(x)$ must also be true. In the concrete case of F-coalgebras, seen as F-realizations, note that R is the edge relation E; then the induction principle (5) is just the standard one associated with a well-founded relation[†].

Assuming (i) and (ii), (iv) can be proved by defining a coalgebra-algebra morphism by induction, using Equation (5). Let F be a type signature with nodes V and let $(A_s, \alpha_s \mid s \in V)$ be an arbitrary F-algebra. Assume the coalgebra C is given in the form of an F-realization $G = (U, E, \mathsf{src}, \mathsf{tgt}, \ell)$. We must define maps $h_s : \ell^{-1}(s) \to A_s$ for $s \in V$ satisfying condition (4). This is equivalent to the following two conditions. Let $s \in V$ and $u \in U$ such that $\ell(u) = s$.

— If s is existential, let d be the unique edge of G with $\mathsf{src}\, d = u$, let $v = \mathsf{tgt}\, d$, and let $e = \ell(d)$. Then

$$h_s(u) = \alpha_s(\mathsf{in}_e(h_{\mathsf{tgt}\, e}(v))) \in A_s.$$

— If s is universal, for each e such that $\mathsf{src}\, e = s$, let d_e be the unique edge with $u = \mathsf{src}\, d_e$ and $\ell(d_e) = e$, and let $v_e = \mathsf{tgt}\, d_e$. Then

$$h_s(u) = \alpha_s(h_{\mathsf{tgt}\, e}(v_e) \mid \mathsf{src}\, e = s) \in A_s.$$

The maps h_s are uniquely defined by these equations due to the well-foundedness of the E-successor relation on G.

Statement (iii) follows as a special case of (iv), since $\mathsf{wf}\, R$ is an F-algebra by Lemma 3.2.

To argue that (iii) implies (i), we observe that under any morphism of F-realizations $C \to \mathsf{wf}\, R$, an infinite path in C would map to an infinite path in $\mathsf{wf}\, R$, which cannot exist by definition, since $\mathsf{wf}\, R$ is well founded. Thus, C must be well founded as well.

For (v) $\Rightarrow$ (iv), suppose that there is a unique *parameterized coalgebra-algebra morphism* from C to any F-algebra. This means simply that for any $\alpha' : FA \times C \to A$, there is a unique h that makes the following diagram commute:

$$\begin{array}{ccc} C & \xrightarrow{\quad h \quad} & A \\ {\scriptstyle \langle \gamma, \mathsf{id} \rangle}\downarrow & & \uparrow{\scriptstyle \alpha'} \\ FC \times C & \xrightarrow[Fh \times \mathsf{id}]{} & FA \times C \end{array} \quad . \tag{6}$$

We wish to show that that there is a unique coalgebra-algebra morphism from C to any F-algebra. Let $\alpha : A \to FA$ be an arbitrary F-algebra and consider $\alpha' = \alpha \circ \pi_1 : FA \times C \to A$.

[†] It might be interesting to note that the proof of the implication (i) $\Rightarrow$ (ii) requires the axiom of dependent choice; see e.g. Dershowitz and Jouannaud (1990); Gries and Schneider (1994).

From the diagram (6), we know that there exists a unique $h : C \to A$ such that

$$h = \alpha \circ \pi_1 \circ (Fh \times \mathsf{id}) \circ \langle \gamma, \mathsf{id} \rangle. \tag{7}$$

We show that h is a coalgebra-algebra morphism from C to A and that it is unique.

$$\begin{aligned} h &= \alpha \circ \pi_1 \circ (Fh \times \mathsf{id}) \circ \langle \gamma, \mathsf{id} \rangle, && \text{diagram (6)} \\ &= \alpha \circ Fh \circ \pi_1 \circ \langle \gamma, \mathsf{id} \rangle, && \pi_1 \text{ is a natural transformation} \\ &= \alpha \circ Fh \circ \gamma && \pi_1 \circ \langle f, g \rangle = f. \end{aligned}$$

For uniqueness, note that any other coalgebra-algebra morphism $g : C \to A$ satisfies:

$$\begin{aligned} g &= \alpha \circ Fg \circ \gamma, && \text{definition of coalgebra-algebra morphism} \\ &= \alpha \circ Fg \circ \pi_1 \circ \langle \gamma, \mathsf{id} \rangle, && \pi_1 \circ \langle f, g \rangle = f \\ &= \alpha \circ \pi_1 \circ (Fg \times \mathsf{id}) \circ \langle \gamma, \mathsf{id} \rangle && \pi_1 \text{ is a natural transformation.} \end{aligned}$$

Hence, using Equation (7), we can conclude $g = h$.

For (iv) $\Rightarrow$ (v), we need the following fact. Let $\gamma : C \to FC$ be an F-coalgebra. Define $G(X) = C \times FX$. If (C, γ) is a well-founded F-coalgebra, then $(C, \langle \gamma, \mathsf{id} \rangle)$ is a well-founded G-coalgebra. If (i) holds for F, then it also holds for G, therefore (iv) holds for G, and (v) follows trivially for F since the diagram (6) for F is a coalgebra-algebra morphism diagram for G. □

Corollary 3.4. The F-coalgebra $\mathsf{wf}\ R$ is (up to isomorphism) the initial F-algebra.

Proof. The structure $\mathsf{wf}\ R$ is an F-algebra by Lemma 3.2. But it is also a well-founded F-coalgebra by definition. By the equivalence of Theorem 3.3(i) and (iv), there is a unique F-algebra morphism from $\mathsf{wf}\ R$ to any F-algebra, thus $\mathsf{wf}\ R$ is initial. □

3.4. *Non-well-founded coalgebras*

In many interesting non-well-founded cases, the diagram (4) does not have a unique solution h. However, for a large class of applications, the codomain A is ordered, and one is interested in the least fixpoint of a monotone map specified by the function definition. This situation was studied in Adámek *et al.* (2006), in which it was shown that under certain conditions on the ordered codomain, functions defined by (4) are preserved by F-coalgebra morphisms. The significance of this property is that a function defined on a non-well-founded coalgebra can be considered a function on the final coalgebra and is independent of the input representation. This covers many examples in which the intended solution is a least fixpoint.

The following result is a minor adaptation of (Adámek *et al.* 2006, Proposition 3.5) to our framework and the proof is similar.

Theorem 3.5 (Adámek *et al.* 2006). Let (A, α) be an ordered F-algebra such that A is a chain-complete and α order-continuous. Let (S, γ) be an F-coalgebra. The construction of the least fixpoint of the map $h \mapsto \alpha \circ Fh \circ \gamma$ is natural in S; that is, if $f : S \to S'$ is an F-coalgebra morphism, then $h_S = h_{S'} \circ f$.

Proof. Let τ_S be the map $h \mapsto \alpha \circ Fh \circ \gamma$ on functions $S \to A$. The assumptions on A and α imply that τ_S is monotone and order-continuous under the pointwise order on $S \to A$. Let $\bot$ be the bottom element of A. The map $\lambda s \in S.\bot$ is the bottom element of $S \to A$. If $f : S \to S'$ is an F-coalgebra morphism, then clearly $\lambda s \in S.\bot = (\lambda s \in S'.\bot) \circ f$, therefore the selection of $\lambda s \in S.\bot$ is natural in S. Moreover, it is easily argued that τ_S is also natural in S; that is, for any $h : S' \to A$, $\tau_S(h \circ f) = \tau_{S'}(h) \circ f$. By induction, $\tau_S^n(\lambda s \in S.\bot)$ is natural in S for all n. By continuity, the least fixpoint is $\sup_n \tau_S^n(\lambda s \in S.\bot)$. The result now follows from the observation that, by definition, suprema of chains are preserved by composition with f on the right. In more detail, recall that the definition of suprema is given for all y by:

$$\sup_n g_n(y) = \left(\sup_n g_n\right)(y). \tag{8}$$

Using the definition twice, one instantiating y to $f(x)$, gives the needed preservation property:

$$\left(\sup_n (g_n \circ f)\right)(x) \overset{(8)}{=} \sup_n g_n(f(x)) \overset{(8)}{=} \left(\sup_n g_n\right)(f(x)) = \left(\left(\sup_n g_n\right) \circ f\right)(x).$$

□

Although Theorem 3.5 covers many interesting non-well-founded situations, there are some that it does not cover. For instance, to define substitution on infinitary λ-terms, the codomain is a coalgebra of infinitary terms, which is not ordered in any natural way. In this case, the solution is unique because it is the image of a coalgebra under the unique homomorphism to the final coalgebra.

4. Well-founded examples

In this section, we present examples of recursive functions that are well founded. The first two, the greatest common divisor of two integers and the towers of Hanoi, already fit the framework of Adámek *et al.* (2007). The others are guaranteed to have a unique solution using the multi-sorted extension to their framework that we have proposed.

4.1. *Integer GCD*

For integers $m, n \geqslant 0$ but not both 0, we would like to compute a triple (g, s, t) such that g is the greatest common divisor (gcd) of m and n and $sm + tn = g$. A recursive definition is

```
let rec gcd m n =
  if n = 0 then (m,1,0) else
  let (q,r) = (m/n, m mod n) in
  let (g,s,t) = gcd n r in
  (g,t,s-t*q)
```

In order to recover this definition as an instantiation of (4), we observe that there is one base case which only needs the value of the first argument – `m`. There is one recursive call that uses one of the arguments – `n` – and takes as second argument `m mod n`. Moreover,

for the final answer after the recursive call we also need we need the value `m/n`. This leads us to the functor $FX = \mathbb{N} + X \times \mathbb{N}$, where the first component of the coproduct will store `m`, and the pair in the second component corresponds to the recursive call and the value `m/n`. We then have the following instantiation of (4):

$$\begin{array}{ccc} \mathbb{N} \times \mathbb{N} & \xrightarrow{h} & \mathbb{N} \times \mathbb{Z} \times \mathbb{Z} \\ {\scriptstyle\gamma}\downarrow & & \uparrow{\scriptstyle\alpha} \\ F(\mathbb{N} \times \mathbb{N}) & \xrightarrow[Fh]{} & F(\mathbb{N} \times \mathbb{Z} \times \mathbb{Z}) \end{array}.$$

Here, the functions γ and α are given by:

$$\gamma(m,n) = \begin{cases} \iota_0(m) & \text{if } n = 0 \\ \iota_1((n, m \bmod n), m/n) & \text{if } n \neq 0 \end{cases} \qquad \begin{array}{l} \alpha(\iota_0(g)) = (g,1,0) \\ \alpha(\iota_1((g,s,t),q)) = (g,t,s-q). \end{array}$$

The theory of recursive coalgebras (Adámek *et al.* 2007) guarantees the existence of a unique function satisfying the diagram.

4.2. *Towers of Hanoi*

Another classic example of a recursive function is the towers of Hanoi. This mathematical game consists of three rods A, B and C and a number of disks of different sizes that can slide on any rod. At the beginning of the game, all disks are on rod A in order of size, smallest on top. The goal of the game is to find a procedure to move all disks to rod B while respecting the following rules:

— only one disk at a time can be moved
— a move consists of removing the upper disk from one of the rods and sliding in onto another rod, on top of other disks that might already be on that rod;
— no disk may be placed on top of a smaller disk.

For n disks, a recursive solution consists in recursively moving $n-1$ disks from the origin rod A to the third rod C, then moving the biggest disk from the origin rod A to the destination rod B, and finally recursively moving $n-1$ disks from the third rod C to the destination rod B. It is given by the following OCaml implementation, where o, d and t are the origin, destination and third rod, respectively:

```
let rec hanoi n o d t =
  if n = 0 then [ ] else
    (hanoi (n-1) o t d) @ [(o,d)] @ (hanoi (n-1) t d o)
```

Let R be the set of rods $\{A, B, C\}$. A move can be represented as an element of R^2 consisting of the origin and the destination of the move. We again have one base case but now two recursive calls. For the base case the result of the function is the empty list. For the other cases, we need the result of two recursive calls plus the pair `(o,d)`. This information is stored in the functor $FX = \mathbb{1} + R^2 \times X^2$: the first component of the coproduct marks the base case (and there is no need to pass on any information since the function will return the empty list); the second component contains the pair `(o,d)`

plus the two placeholders for the recursive calls – X^2. All in all, this gives the following instantiation of Equation (4):

$$\begin{array}{ccc} \mathbb{N} \times R^3 & \xrightarrow{\quad h \quad} & \left(R^2\right)^* \\ {\scriptstyle\gamma}\downarrow & & \uparrow{\scriptstyle\alpha} \\ \mathbb{1} + R^2 \times (\mathbb{N} \times R^3) \times (\mathbb{N} \times R^3) & \xrightarrow[Fh]{} & \mathbb{1} + R^2 \times \left(R^2\right)^* \times \left(R^2\right)^* \end{array}.$$

Here, γ and α are given by:

$$\gamma(n,o,d,t) = \begin{cases} \iota_0(), & \text{if } n = 0, \\ \iota_1((o,d),(n-1,o,t,d),(n-1,t,d,o)), & \text{if } n \neq 0 \end{cases}$$

$$\alpha(\iota_0()) = \varepsilon \qquad \alpha(\iota_1((o,d),b,e)) = b \cdot [(o,d)] \cdot e.$$

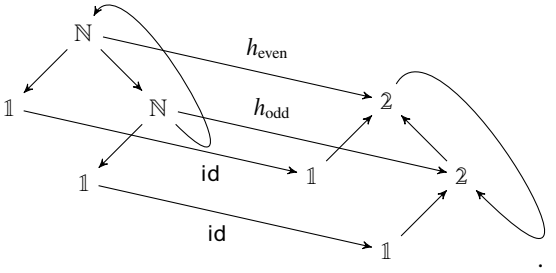
The theory of recursive coalgebras (Adámek *et al.* 2007) guarantees the existence of a unique function satisfying the diagram.

4.3. *Mutually recursive functions:* `even-odd`

This subsection illustrates how our generalization to multi-sorted signatures handles mutually recursive functions in a symmetric way. A very simple example is the definition of the even and odd predicates on natural numbers.

```
let rec even n = if n = 0 then true else odd (n-1)
and odd n = if n = 0 then false else even (n-1)
```

We can depict the recursion graphically with the following diagram:



This can be viewed as an endofunctor $F : \mathsf{Set}^V \to \mathsf{Set}^V$, where $V = \{\mathsf{even}, \mathsf{odd}\}$. The functor is defined by: $F(A,B) = (\mathbb{1} + B, \mathbb{1} + A)$ and if $g : A \to A'$ and $h : B \to B'$, then $F(g,h) = (\mathsf{id} + h, \mathsf{id} + g) : F(A,B) \to F(A',B')$.

An F-coalgebra is a pair $((C,D),\gamma)$, where $\gamma : (C,D) \to F(C,D)$ is a morphism in the underlying category Set^V; that is,

$$\gamma = (\gamma_{\mathsf{even}}, \gamma_{\mathsf{odd}}) : (C,D) \to (\mathbb{1} + D, \mathbb{1} + C),$$

where $\gamma_{\mathsf{even}} : C \to \mathbb{1} + D$ and $\gamma_{\mathsf{odd}} : D \to \mathbb{1} + C$. Similarly, an F-algebra is a pair $((A,B),\alpha)$, where $\alpha : F(A,B) \to (A,B)$ is a morphism in Set^V; that is,

$$\alpha = (\alpha_{\mathsf{even}}, \alpha_{\mathsf{odd}}) : (\mathbb{1} + B, \mathbb{1} + A) \to (A,B),$$

where $\alpha_{\mathsf{even}} : \mathbb{1} + B \to A$ and $\alpha_{\mathsf{odd}} : \mathbb{1} + A \to B$.

An F-algebra-coalgebra morphism $h : ((C, D), \gamma) \to ((A, B), \alpha)$ is a map $h = (h_{\mathsf{even}}, h_{\mathsf{odd}}) : (C, D) \to (A, B)$ such that the following diagram commutes:

$$\begin{array}{ccc} (C,D) & \xrightarrow{(h_{\mathsf{even}}, h_{\mathsf{odd}})} & (A,B) \\ {\scriptstyle (\gamma_{\mathsf{even}}, \gamma_{\mathsf{odd}})}\downarrow & & \uparrow{\scriptstyle (\alpha_{\mathsf{even}}, \alpha_{\mathsf{odd}})} \\ (\mathbb{1}+D, \mathbb{1}+C) & \xrightarrow[(\mathsf{id}+h_{\mathsf{odd}}, \mathsf{id}+h_{\mathsf{even}})]{} & (\mathbb{1}+B, \mathbb{1}+A) \end{array}.$$

In our application, we have $A = B = \mathbb{2}$ and $C = D = \mathbb{N}$, with

$$\gamma_{\mathsf{even}}(n) = \gamma_{\mathsf{odd}}(n) = \begin{cases} \imath_0() & \text{if } n = 0 \\ \imath_1(n-1) & \text{if } n > 0 \end{cases}$$

$$\alpha_{\mathsf{even}}(\imath_0()) = \mathbf{1} \qquad \alpha_{\mathsf{odd}}(\imath_0()) = \mathbf{0} \qquad \alpha_{\mathsf{even}}(\imath_1(b)) = \alpha_{\mathsf{odd}(\imath_1(b))} = b.$$

4.4. *Ackermann function*

The Ackermann function

$$A(0,n) = n+1 \qquad A(m+1,0) = A(m,1) \qquad A(m+1,n+1) = A(m, A(m+1,n)) \tag{9}$$

is a notoriously fast-growing function that also fits into our general scheme (although one should not try to compute it!). This example is quite interesting, because at first glance it seems not to fit the scheme (4) because of the nested recursive call in the third clause. However, a key insight comes from the termination proof, which is done by induction on the well-founded lexicographic order on $\mathbb{N} \times \mathbb{N}$ with m as the more significant parameter. We see that we can break the definition into two stages, both higher-order.

Rewriting $A(m,n)$ as $A_m(n)$, we have that Equation (9) is equivalent to

$$A_0 = \lambda n.n+1 \qquad\qquad A_{m+1} = \lambda n.A_m^{n+1}(1),$$

where f^n denotes the n-fold composition of f with itself:

$$f^0 = \lambda n.n \qquad\qquad f^{n+1} = f \circ f^n.$$

The outermost stage computes $m \mapsto A_m$. The corresponding diagram is

$$\begin{array}{ccc} \mathbb{N} & \xrightarrow{A} & \mathbb{N}^{\mathbb{N}} \\ {\scriptstyle \gamma}\downarrow & & \uparrow{\scriptstyle \alpha} \\ \mathbb{1}+\mathbb{N} & \xrightarrow[\mathsf{id}_{\mathbb{1}}+A]{} & \mathbb{1}+\mathbb{N}^{\mathbb{N}} \end{array}$$

where

$$\gamma(0) = \imath_0() \qquad \gamma(m+1) = \imath_1(m) \qquad \alpha(\imath_0()) = \lambda n.n+1 \qquad \alpha(\imath_1(f)) = \lambda n.f^{n+1}(1).$$

In turn, the function α is defined in terms the iterated composition function $\mathsf{comp}(n, f) = f^n$, defined for functions on a generic domain D by:

$$\begin{array}{ccc} \mathbb{N} \times D^D & \xrightarrow{\mathsf{comp}} & D^D \\ {\scriptstyle\gamma'}\downarrow & & \uparrow{\scriptstyle\alpha'} \\ F(\mathbb{N} \times D^D) & \xrightarrow[F(\mathsf{comp})]{} & F(D^D) \end{array}$$

where $FX = \mathbb{1} + D^D \times X$ and

$$\gamma'(0, f) = \iota_0() \qquad \gamma'(n+1, f) = \iota_1(f, (n, f)) \qquad \alpha'(\iota_0()) = \mathsf{id}_D \qquad \alpha'(\iota_1(f, g)) = f \circ g.$$

Hence, the Ackermann function (somewhat surprisingly to us) also fits the scheme (4).

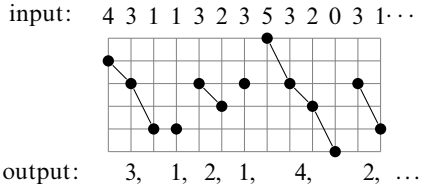
5. Non-well-founded examples

We provided many examples of non-well-founded functions in Jeannin *et al.* (2012, 2013), including probabilistic protocols, p-adic numbers, and a fairly substantial example involving abstract interpretation. Here we present a few more. We will also present an example involving the set of elements in an infinite list in Section 6.

5.1. *Descending sequences*

As one of the simplest nontrivial coinductive datatypes, *streams* offer the ideal playground to test new theories. We present an example on streams of natural numbers $\mathbb{N}^\omega$. The following example, taken from a talk by Capretta (Capretta 2007), has a unique solution, but does not fit the existing theory of well-founded coalgebras (Adámek *et al.* 2007) or our generalization presented here, nor does it fit the theory of corecursive algebras (Capretta *et al.* 2009).

The goal is to produce from a given stream of natural numbers another stream of natural numbers containing the lengths of the maximal strictly descending subsequences of the input stream. An example is shown in the following figure, where the input stream is depicted in a grid to show the order of elements.



Here is a simple recursive definition of the function in CoCaml (see Section 6), where the `constructor` solver builds a new stream:

```
let descending arg =
  let corec[constructor] descending_aux (n, i :: j :: t) =
    if i > j then descending_aux (n+1, j :: t)
    else n :: descending_aux (1, j :: t) in
  descending_aux (1, arg)
```

This definition corresponds to the following instantiation of Equation (4):

$$\begin{array}{ccc} \mathbb{N} \times \mathbb{N}^\omega & \xrightarrow{\quad h \quad} & \mathbb{N}^\omega \\ {\scriptstyle\gamma}\downarrow & & \uparrow{\scriptstyle\alpha} \\ \mathbb{N} \times \mathbb{N}^\omega + \mathbb{N} \times (\mathbb{N} \times \mathbb{N}^\omega) & \xrightarrow[h + \mathsf{id}_{\mathbb{N}} \times h]{} & \mathbb{N}^\omega + \mathbb{N} \times \mathbb{N}^\omega \end{array}$$

where $FX = X + \mathbb{N} \times X$ and

$$\gamma(n, i :: j :: t) = \begin{cases} \iota_0(n+1, j :: t) & \text{if } i > j \\ \iota_1(n, (1, j :: t)) & \text{otherwise} \end{cases} \qquad \alpha(\iota_0(s)) = s \qquad \alpha(\iota_1(n, s)) = n :: s.$$

The `constructor` solver is one of the built-in solvers in CoCaml. It is used to construct a solution that is an element of a final coalgebra. In such situations, the solution is unique, but not necessarily the least fixpoint of any monotone map, as the codomain is not ordered.

5.2. *Alternating turing machines and* IND *programs*

Alternating Turing machines (see e.g. Papadimitriou (1993); Sipser (2006)) are like nondeterministic Turing machines, except they can make universal ($\wedge$) branches as well as existential ($\vee$) branches. An existential configuration is accepting if at least one of its successor configurations is accepting, whereas a universal configuration is accepting if all of its successor configurations are accepting.

Formally, the semantics of alternating Turing machines is described in terms of an inductive labelling of machine configurations C with either $\mathbf{0}$ (rejecting), $\mathbf{1}$ (accepting) or $\bot$ (undetermined). In the present framework, the function γ would give the set of successor configurations and the labelling of the state as either existential or universal, and α would tell how to label configurations $\mathbf{0}$, $\mathbf{1}$, or $\bot$ inductively up the computation tree. Formally, α gives the infimum for universal configurations and supremum for existential configurations in three-valued Kleene logic $\mathbb{3} = \{\mathbf{0}, \bot, \mathbf{1}\}$ with ordering $\mathbf{0} \leqslant \bot \leqslant \mathbf{1}$.

$$\begin{array}{ccc} C & \xrightarrow{\quad h \quad} & \mathbb{3} \\ {\scriptstyle\gamma}\downarrow & & \uparrow{\scriptstyle\alpha} \\ \mathbb{2} \times \mathcal{P}_{\text{fin}}(C) & \xrightarrow[\mathsf{id}_{\mathbb{2}} + \mathcal{P}_{\text{fin}}(h)]{} & \mathbb{2} \times \mathcal{P}_{\text{fin}}(\mathbb{3}) \end{array}$$

The canonical solution is defined to be the least fixpoint with respect to a different order, namely the Scott order $\bot \sqsubseteq \mathbf{0}$, $\bot \sqsubseteq \mathbf{1}$. This example is interesting, because it is a case in which α is not strict; for example, a universal configuration can be labelled $\mathbf{0}$ as soon as one of its successors is known to be labelled $\mathbf{0}$, regardless of the labels of the other successors.

A similar model is the IND programming language for the inductive sets (Harel and Kozen 1984). An IND program consists of a sequence of labelled statements of three kinds: universal and existential assignment ($x := \forall$ and $x := \exists$, respectively), conditional test (if $s = t$ then ℓ_1 else ℓ_2) and halting (accept, reject). The semantics of IND

programs is very similar to alternating Turing machines. The statement $x := \exists$ assigns a nondeterministically chosen element of the domain to the variable x, spawning as many new processes as the cardinality of the domain. The computation from that point accepts if at least one of the newly spawned processes accepts. Similarly, the statement $x := \forall$ accepts if all the newly spawned processes accept. Thus, the semantics is the same as alternating Turing machines, except that the branching degree is equal to the cardinality of the domain of computation. IND programs accept exactly the inductively definable sets, which over $\mathbb{N}$ are exactly the Π^1_1 sets.

6. CoCaml

Along with our study characterizing the existence and uniqueness of solutions of diagram (4), we also became interested in situations in which solutions exist but are not unique. There are many interesting such cases, as our non-well-founded examples have shown. Often there is a desired solution to (4), but it is not the one computed by the standard semantics of recursion. We wanted to provide language constructs for the programmer to specify alternative solutions in those cases. This led to the design of an extension of OCaml called CoCaml (CoCaml 2012; Jeannin *et al.* 2012, 2013). The language is described in more detail there, but we would like to give a flavor of it in this section.

We provide some motivation using a function over streams, coded in OCaml as infinite lists. In OCaml, the type of finite and infinite lists is built in. The empty list is written `[]`, and the list with head `h` and tail `t` is written `h :: t`. Infinite objects of this type can be defined using the `let rec` construct. For example,

```
let rec ones = 1 :: ones
let rec alt = 1 :: 2 :: alt
```

The first example defines the infinite sequence of ones $1, 1, 1, 1, \ldots$ and the second the sequence $1, 2, 1, 2, \ldots$.

The `let rec` construct allows us to build only *regular* lists, that is, those that are ultimately periodic. Such lists always have a finite representation in memory. The coinductive elements we consider are always regular; that is, they have a finite but possibly cyclic representation. This is different from a setting in which infinite elements are represented lazily and are computed on the fly.

Although the `let rec` construct allows us to specify (finite representations of) infinite structures, further investigation reveals a major shortcoming. For example, suppose we wanted to define a function that, given an infinite list, returns the set of its elements. For the lists `ones` and `alt`, the function should return the sets $\{1\}$ and $\{1, 2\}$, respectively. One would like to write a function definition using the obvious equations which pattern-match on the two constructors of the `list` datatype:

```
let set l = match l with
| [ ] -> [ ]
| h :: t -> insert h (set t)
```

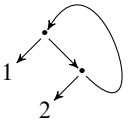
where `insert` inserts an element in a set, represented say by a sorted finite list without duplicates. However, this function will not halt in OCaml on the lists `ones` and `alt`,

even though it is clear what the answers should be. Note that this is not a corecursive definition, as we are not asking for a greatest solution or a unique solution in a final coalgebra, but rather a least solution in a different ordered domain from the one provided by the standard semantics of recursive functions. The standard semantics of recursive functions gives us the least solution in a domain with bottom element $\perp$ representing nontermination, whereas we would like the least solution in a different CPO, namely $(\mathcal{P}(\mathbb{Z}), \subseteq)$ with bottom element $\varnothing$.

The CoCaml language extends OCaml with a construct that allows functions defined by equations, like the one above, to be supplied with an extra parameter, namely a solver for the given equations. For instance, the example above would be almost the same in CoCaml:

```
let corec[iterator([ ])] set l = match l with
| [ ] -> [ ]
| h :: t -> insert h (set t)
```

The construct `corec` with the parameter `iterator([])` specifies to the compiler that the equations above should be solved using the built-in `iterator` solver—in this case a least fixpoint computation – starting with the initial element `[]`. For the infinite list `alt`, which can abstractly be thought of as the circular structure



the compiler will generate two equations:

```
set(x) = insert 1 (set(y))
set(y) = insert 2 (set(x))
```

then solve them using the specified solver `iterator`, which will produce the intended set $\{1, 2\}$.

CoCaml has a number of built-in solvers (`iterator`, `constructor` and a Gaussian elimination solver `gaussian`), as well as an interface for programmers to create their own solvers; see Jeannin *et al.* (2012, 2013) and the CoCaml project website (CoCaml 2012) for details. The solver `constructor`, which we already mentioned in the descending sequence example of the previous section, is particularly interesting, since it enables the definition of functions whose codomain is a given final coalgebra.

7. Discussion

In this paper, we have presented the origins of our work on bringing coinduction to a functional language in the form of effective language constructs.

The work in the present paper and related implementation papers (Jeannin *et al.* 2012, 2013) was inspired by work on recursive coalgebras (Adámek *et al.* 2007) and Elgot algebras (Adámek *et al.* 2006). We have extended and clarified the results in Adámek *et al.* (2007) by providing a different proof that works on a larger class of functors. Our generalization handles multi-sorted signatures and mutually recursive functions in a

symmetric way and is not restricted to finitary functors. We have also provided several examples of functions defined using this scheme, as well as non-well-founded examples that do not have a unique solution but still have a canonical solution. Finally, we have briefly described our work on programming language constructs to allow the programmer to choose alternative solution methods when the standard semantics of recursion would not halt.

The theoretical results of Adámek *et al.* (2007) and the results of this paper are concerned with the properties of the domain C ensuring unique solutions to the diagram (4). The motivation for recursive coalgebras stems from the seminal work of Osius (1974) on coalgebras of the powerset functor, generalized in Paul Taylor's monograph (Taylor 2009). Capretta *et al.* (2009) studied the dual problem of characterizing properties of the codomain A ensuring this property. The work of Adámek, Milius, and Velebil on Elgot algebras (Adámek *et al.* 2006) is relevant to our work on recursive definitions that do not have unique solutions. Elgot algebras provide specified canonical solutions rather than unique ones. The canonical solutions must satisfy two axioms, the first ensuring that solutions are independent of the representation of the input and are thus well-defined on a final coalgebra, and the second that allows multiple fixpoints to be parameterized and computed sequentially. The latter property gives an alternative approach to mutually recursive functions. There is also some related work of a more practical nature (Hirschowitz *et al.* 2003; Sperber and Thiemann 1998; Syme 2006; Trancón y Widemann 2011) that we discuss in Jeannin *et al.* (2012, 2013).

Acknowledgments

Thanks to Stefan Milius for stimulating discussions. We are grateful for detailed comments of Ernst Doberkat and Alexander Kurz which helped us improving the presentation.

Financial support

The third author was partially supported by the Dutch Research Foundation (NWO), project numbers 639.021.334 and 612.001.113. The second author was supported by the National Security Agency.

Conflict of interest

None.

References

Adámek, J., Lücke, D. and Milius, S. (2007). Recursive coalgebras of finitary functors. *Theoretical Informatics and Applications* **41** (4) 447–462.

Adámek, J., Milius, S. and Velebil, S. (2006). Elgot algebras. *Logical Methods in Computer Science* **2** (5:4) 1–31.

Capretta, V. (2007). An introduction to corecursive algebras. `http://www.cs.ru.nl/~venanzio/publications/brouwer_seminar_4_12_2007.pdf`.

Capretta, V., Uustalu, T. and Vene, V. (2009). Corecursive algebras: A study of general structured corecursion. In: Vinicius, M., Oliveira, M. and Woodcock, J. (eds.) Formal Methods: Foundations and Applications, 12th Brazilian Symp. Formal Methods (SBMF 2009). *Lecture Notes in Computer Science* **5902,** Springer, Berlin, 84–100.

CoCaml project. (December 2012). `http://www.cs.cornell.edu/Projects/CoCaml/`.

Dershowitz, N. and Jouannaud, J.-P. (1990). Rewrite systems. In: van Leeuwen, J. (ed.) *Handbook of Theoretical Computer Science*, Formal Models and Semantics, volume B, chapter 6, Elsevier, 243–320.

Gries, D. and Schneider, F.B. (1994). *A Logical Approach to Discrete Math*. Springer-Verlag.

Harel, D. and Kozen, D. (1984). A programming language for the inductive sets, and applications. *Information and Control* **63** (1–2) 118–139.

Hirschowitz, T., Leroy, X. and Wells, J.B. (2003). Compilation of extended recursion in call-by-value functional languages. In: *PPDP 2003* 160–171.

Jeannin, J.-B., Kozen, D. and Silva, A. (December 2012). CoCaml: Programming with coinductive types. Technical Report `http://hdl.handle.net/1813/30798`, Computing and Information Science, Cornell University.

Jeannin, J.-B., Kozen, D. and Silva, A. (March 2013). Language constructs for non-well-founded computation. In: Felleisen, M. and Gardner, P. (eds.) 22nd European Symposium on Programming (ESOP 2013). *Lecture Notes in Computer Science* **7792**, Springer, Rome, Italy, 61–80.

Kozen, D. (May 2011). Realization of coinductive types. In: Mislove, M. and Ouaknine, J. (eds.) Proceedings of the 27th Conf. Math. Found. Programming Semantics (MFPS XXVII), Pittsburgh, PA. *Elsevier Electronic Notes in Theoretical Computer Science* 148–155.

Lambek, J. (1968). A fixpoint theorem for complete categories. *Mathematische Zeitschrift* **103** (2) 151–161.

Mac Lane, S. (1971). *Categories for the Working Matematician*. Springer.

Osius, G. (1974). Categorical set theory: A characterization of the category of sets. *Journal of Pure and Applied Algebra* **4** 79–119.

Papadimitriou, C. (1993). *Computational Complexity*. Addison Wesley.

Sipser, M. (2006). *Introduction to the Theory of Computation*, 2nd edition, PWS Publishing.

Sperber, M. and Thiemann, P. (September 1998). ML and the address operator. In: *Proceedings of the ACM SIGPLAN Workshop on ML* 152–153.

Syme, D. (2006). Initializing mutually referential abstract objects: The value recursion challenge. *Electronic Notes in Theoretical Computer Science*, **148** (2) 3–25.

Taylor, P. (1999). *Practical Foundations of Mathematics*. Cambridge University Press.

Trancón y Widemann, B. (August 2011). Coalgebraic semantics of recursion on circular data structures. In: Cirstea, C., Seisenberger, M. and Wilkinson, T. (eds.) *CALCO Young Researchers Workshop (CALCO-jnr 2011)* 28–42.